main_xgb

Minzhi Zhang mz2787

10/27/2020

```r
if(!require("EBImage")){
  install.packages("BiocManager")
  BiocManager::install("EBImage")
}
if(!require("R.matlab")){
  install.packages("R.matlab")
}
if(!require("readxl")){
  install.packages("readxl")
}

if(!require("dplyr")){
  install.packages("dplyr")
}
if(!require("readxl")){
  install.packages("readxl")
}

if(!require("ggplot2")){
  install.packages("ggplot2")
}

if(!require("caret")){
  install.packages("caret")
}

if(!require("glmnet")){
  install.packages("glmnet")
}

if(!require("WeightedROC")){
  install.packages("WeightedROC")
}

if(!require("WeightedROC")){
  install.packages("WeightedROC")
}

if(!require("xgboost")){
  install.packages("xgboost")
}
```

```r
if(!require("DMwR")){
  install.packages("DMwR")
}

library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(xgboost)
library(DMwR)
```

Step 0 set work directories

Provide directories for training images. Training images and Training fiducial points will be in different subfolders.

```r
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_image_dir <- paste(train_dir, "images/", sep="")
train_pt_dir <- paste(train_dir,  "points/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (T/F) reweighting the samples for training set
- (T/F) oversampling the samples for training set using SMOTE
- (number) K, the number of CV folds
- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

```r
run.cv <- TRUE # run cross-validation on the training set
sample.reweight <- TRUE # run sample reweighting in model training
smote <- TRUE # run SMOTE on in model training
K <- 5  # number of CV folds
run.feature.train <- TRUE # process features for training set
run.test <- TRUE # run evaluation on an independent test set
run.feature.test <- TRUE # process features for test set
```

Step 2: import data and train-test split

```r
#train-test split
set.seed(2020)
info <- read.csv(train_label_path)
n <- nrow(info)
n_train <- round(n*(4/5), 0)
```

```
train_idx <- sample(info$Index, n_train, replace = F)
test_idx <- setdiff(info$Index, train_idx)
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
readMat.matrix <- function(index){
     return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1]],0))
}

#load fiducial points
n_files <- length(list.files(train_image_dir))
fiducial_pt_list <- lapply(1:n_files, readMat.matrix)
save(fiducial_pt_list, file="../output/fiducial_pt_list.RData")
```

Step 3: construct features and responses

```
source("../lib/feature.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(fiducial_pt_list, train_idx))
  save(dat_train, file="../output/feature_train.RData")
}else{
  load(file="../output/feature_train.RData")
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list, test_idx))
  save(dat_test, file="../output/feature_test.RData")
}else{
  load(file="../output/feature_test.RData")
}
```

```
# oversampling data using SMOTE method
library(DMwR)
set.seed(2020)
tm_smote <- system.time(train_smote <- SMOTE(label ~ ., dat_train, perc.over = 200, k = 5,
                                             perc.under = 150))
tm_smote
```

```
##    user  system elapsed
##  52.822   7.169  60.239
```

Step 4: Train a classification model with training features and responses

```
source("../lib/train.R")
source("../lib/test.R")
```

Model selection with cross-validation

Cross Validation with Reweighting sample

```
# cross validation with reweighting data, tuning nrounds and max_depth
source("../lib/cross_validation.R")

nrounds_list <- seq(20, 100, 20)
max_depth_list <- c(10, 20)
run.cv <- FALSE
if(run.cv){
  res_cv_rw <- cv.function.xgb(dat_train, 5, reweight = TRUE, smote = FALSE,
                               nrounds_list, max_depth_list)
  save(res_cv_rw, file="../output/res_cv_rw.RData")
}else{
  load("../output/res_cv_rw.RData")
}

res_cv_rw
```

```
## $mean_error
##           [,1]      [,2]
## [1,] 0.3433302 0.3636274
## [2,] 0.3565976 0.3589426
## [3,] 0.3707250 0.3487006
## [4,] 0.3597258 0.3439876
## [5,] 0.3427746 0.3422774
##
## $mean_AUC
##           [,1]      [,2]
## [1,] 0.7837712 0.7768042
## [2,] 0.7899824 0.7978492
## [3,] 0.8002050 0.8092118
## [4,] 0.8041076 0.8001706
## [5,] 0.8012920 0.8074034
```

Cross Validation with SMOTE

```
# cross validation with SMOTE, tuning nrounds and max_depth
source("../lib/cross_validation.R")

nrounds_list <- seq(20, 100, 20)
max_depth_list <- c(10, 20)
run.cv <- FALSE
if(run.cv){
  res_cv_sm <- cv.function.xgb(dat_train, 5, reweight = FALSE, smote = TRUE,
                               nrounds_list, max_depth_list)
  save(res_cv_sm, file="../output/res_cv_sm.RData")
}else{
  load("../output/res_cv_sm.RData")
}

res_cv_sm
```

```
## $mean_error
##           [,1]      [,2]
## [1,] 0.0921242 0.0959984
## [2,] 0.0963390 0.0798202
## [3,] 0.0847360 0.0843836
## [4,] 0.0826272 0.0840486
## [5,] 0.0794810 0.0872006
##
## $mean_AUC
##           [,1]      [,2]
## [1,] 0.9732210 0.9693970
## [2,] 0.9742660 0.9765240
## [3,] 0.9776792 0.9777940
## [4,] 0.9801844 0.9762790
## [5,] 0.9785240 0.9765744
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

Reweighting sample

```
# source("../lib/train.R")
train_label <- as.numeric(levels(dat_train$label))[dat_train$label]
weight_train <- rep(NA, length(train_label))
for (v in unique(train_label)){
  weight_train[train_label == v] = 0.5 * length(train_label) / length(train_label[train_label == v])
}


train_xgb <- xgb.DMatrix(as.matrix(dat_train[, -6007]),
                           label = train_label,
                           weight = weight_train)
tm_train_rw <- system.time(fit_train_rw <- train.xgb(train_xgb, nrounds = 100, max_depth = 20))

save(fit_train_rw, file="../output/fit_train_rw.RData")
```

SMOTE

```
train_label <- as.numeric(levels(train_smote$label))[train_smote$label]
train_xgb <- xgb.DMatrix(as.matrix(train_smote[, -6007]), label = train_label)

tm_train_sm <- system.time(fit_train_sm <- train.xgb(train_xgb, nrounds = 80, max_depth = 10))

save(fit_train_sm, file="../output/fit_train_sm.RData")
```

Step 5: Run test on test images

Reweighting sample

```
test_label <- as.numeric(levels(dat_test$label))[dat_test$label]
weight_test <- rep(NA, length(test_label))
for (v in unique(test_label)){
  weight_test[test_label == v] = 0.5 * length(test_label) / length(test_label[test_label == v])
}

tm_test_rw = NA
```

```
feature_test <- as.matrix(dat_test[, -6007])
if(run.test){
  load(file="../output/fit_train_rw.RData")
  test_label <- as.numeric(levels(dat_test$label))[dat_test$label]
  test_xgb <- xgb.DMatrix(as.matrix(dat_test[, -6007]),
                          label = test_label,
                          weight = weight_test)
  tm_test_rw <- system.time({prob_pred <- test.xgb(fit_train_rw, test_xgb)[1];
                           label_pred <- test.xgb(fit_train_rw, test_xgb)[2]})
}
```

- evaluation

```
## reweight the test data to represent a balanced label distribution
label_test <- as.integer(dat_test$label)-1
weight_test <- rep(NA, length(label_test))
for (v in unique(label_test)){
  weight_test[label_test == v] = 0.5 * length(label_test) / length(label_test[label_test == v])
}

accu <- sum(weight_test * as.numeric(unlist(label_pred)) == label_test)/sum(weight_test)
tpr.fpr <- WeightedROC(as.numeric(unlist(prob_pred)), label_test, weight_test)
auc <- WeightedAUC(tpr.fpr)


cat("The accuracy of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is", accu*100, "%
cat("The AUC of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is", auc, ".\n")
```

```
## The accuracy of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is 78.33333 %.
## The AUC of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is 0.8476575 .
```

- confusion matrix

```
label_pred <- as.numeric(unlist(label_pred))
cf_mat <- table(label_pred, label_test)
cf_mat

TN <- cf_mat[1,1]
FP <- cf_mat[2,1]
FN <- cf_mat[1,2]
TP <- cf_mat[2,2]

Precision <- TP/(TP+FP)
Sensitivity <- TP/(TP+FN)
Specificity <- TN/(TN+FP)
F_score <- 2*Precision*Sensitivity/(Precision+Sensitivity)

cat("The Precision of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is", Precision*10
cat("The Sensitivity of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is", Sensitivit
cat("The Specificity of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is", Specificit
cat("The F score of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is", F_score*100,
```

```
##           label_test
## label_pred   0   1
##          0 470  65
##          1  19  46
## The Precision of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is 70.76923 %.
## The Sensitivity of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is 41.44144 %.
## The Specificity of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is 96.11452 %.
## The F score of model XGB with reweighting sample, nrounds = 100, max_depth = 20 is 52.27273 %.
```

SMOTE

```
# SMOTE
source("../lib/test.R")
tm_test = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test){
  load(file="../output/fit_train_sm.RData")
  test_label <- as.numeric(levels(dat_test$label))[dat_test$label]
  test_xgb <- xgb.DMatrix(as.matrix(dat_test[, -6007]),
                          label = test_label)
  tm_test_sm <- system.time({prob_pred <- test.xgb(fit_train_sm, test_xgb)[1];
                          label_pred <- test.xgb(fit_train_sm, test_xgb)[2]})
}
```

- evaluation

```
# SMOTE
label_test <- as.integer(dat_test$label)-1


accu <- sum(as.numeric(unlist(label_pred)) == label_test)/length(label_test)
tpr.fpr <- WeightedROC(as.numeric(unlist(prob_pred)), label_test)
auc <- WeightedAUC(tpr.fpr)


cat("The accuracy of model XGB with SMOTE, nrounds = 80, max_depth = 10 is", accu*100, "%.\n")
cat("The AUC of model XGB with SMOTE, nrounds = 80, max_depth = 10 is", auc, ".\n")
```

```
## The accuracy of model XGB with SMOTE, nrounds = 80, max_depth = 10 is 81.33333 %.
## The AUC of model XGB with SMOTE, nrounds = 80, max_depth = 10 is 0.8177564 .
```

- confusion matrix

```
label_pred <- as.numeric(unlist(label_pred))
cf_mat <- table(label_pred, label_test)

TN <- cf_mat[1,1]
FP <- cf_mat[2,1]
FN <- cf_mat[1,2]
TP <- cf_mat[2,2]

Precision <- TP/(TP+FP)
Sensitivity <- TP/(TP+FN)
Specificity <- TN/(TN+FP)
F_score <- 2*Precision*Sensitivity/(Precision+Sensitivity)

cat("The Precision of model XGB with SMOTE, nrounds = 100, max_depth = 20 is", Precision*100, "%.\n")
cat("The Sensitivity of model XGB with SMOTE, nrounds = 100, max_depth = 20 is", Sensitivity*100, "%.\n
cat("The Specificity of model XGB with SMOTE, nrounds = 100, max_depth = 20 is", Specificity*100, "%.\n
cat("The F score of model XGB with SMOTE, nrounds = 100, max_depth = 20 is", F_score*100, "%.\n")
```

```
## The Precision of model XGB with SMOTE, nrounds = 100, max_depth = 20 is 49.54955 %.
## The Sensitivity of model XGB with SMOTE, nrounds = 100, max_depth = 20 is 49.54955 %.
## The Specificity of model XGB with SMOTE, nrounds = 100, max_depth = 20 is 88.54806 %.
## The F score of model XGB with SMOTE, nrounds = 100, max_depth = 20 is 49.54955 %.
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

Reweighting Sample

```
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training model=", tm_train_rw[1], "s \n")
cat("Time for testing model=", tm_test_rw[1], "s \n")
```

```
## Time for constructing training features= 1.912 s
## Time for constructing testing features= 0.201 s
## Time for training model= 218.753 s
## Time for testing model= 0.244 s
```

SMOTE

```
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training model=", tm_train_sm[1], "s \n")
cat("Time for testing model=", tm_test_sm[1], "s \n")
```

```
## Time for constructing training features= 1.912 s
## Time for constructing testing features= 0.201 s
## Time for training model= 284.021 s
## Time for testing model= 0.205 s
```